Computational Optimization and Applications, 2, (1993), 229–260

A Generic Auction Algorithm for the Minimum Cost Network Flow Problem

DIMITRI P. BERTSEKAS
Department of Electrical Engineering and Computer Science, M.I.T., Cambridge, MA 02139

DAVID A. CASTAÑON
Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215

Abstract. In this paper we broadly generalize the assignment auction algorithm to solve linear minimum cost network flow problems. We introduce a generic algorithm, which contains as special cases a number of known algorithms, including the ϵ-relaxation method, and the auction algorithm for assignment and for transportation problems. The generic algorithm can serve as a broadly useful framework for the development and the complexity analysis of specialized auction algorithms that exploit the structure of particular network problems. Using this framework, we develop and analyze two new algorithms, an algorithm for general minimum cost flow problems, called network auction, and an algorithm for the k node-disjoint shortest path problem.

Keywords: Optimization, network programming, auction, transportation.

1. Introduction

In this paper we discuss algorithms for solution of the classical minimum cost network flow problem involving a directed graph with node set $\mathcal{N}$ and arc set $\mathcal{A}$. Each arc (i, j) has a cost coefficient a_{ij}. Letting x_{ij} be the flow of the arc (i, j), the problem is

$$\text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} \tag{LNF}$$

subject to

$$\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = s_i, \quad \forall\, i \in \mathcal{N}, \tag{1}$$

$$b_{ij} \le x_{ij} \le c_{ij}, \quad \forall\, (i, j) \in \mathcal{A}, \tag{2}$$

where a_{ij}, b_{ij}, c_{ij}, and s_i are given integers.

We denote by x the vector with elements x_{ij}, $(i, j) \in \mathcal{A}$. We refer to b_{ij} and c_{ij}, and the interval $[b_{ij}, c_{ij}]$ as the *flow bounds* and the *feasible flow range* of arc (i, j), respectively. We refer to s_i as the *supply* of node i. The constraints (1) and (2) are called the *conservation of flow constraints* and the *capacity constraints*, respectively. A flow vector satisfying both of these constraints is called *feasible*, and if it satisfies just the capacity constraints, it is called *capacity-feasible*. If there exists at least one feasible flow vector, problem (LNF) is called *feasible*

and otherwise it is called *infeasible*. For a given flow vector x, the *divergence* of node i is defined to be the total flow coming out of i minus the total flow coming into i,

$$y_i = \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji}.$$

The *surplus* of node i is defined as the difference between the supply and the divergence of i,

$$g_i = s_i - y_i. \tag{3}$$

We assume that there exists at most one arc in each direction between any pair of nodes, but this assumption is made for notational convenience and can be easily dispensed with. We denote the numbers of nodes and arcs by N and A, respectively. We also denote by C the maximum absolute value of the cost coefficients,

$$C = \max_{(i,j)\in\mathcal{A}} |a_{ij}|. \tag{4}$$

We use the following well-known dual problem to (LNF), which involves a price variable p_i for each node i:

$$\begin{aligned} &\text{maximize } q(p) \\ &\text{subject to no constraint on } p, \end{aligned} \tag{5}$$

where p is the vector with elements p_i, and the dual function q given by

$$q(p) = \sum_{(i,j)\in\mathcal{A}} q_{ij}(p_i - p_j) + \sum_{i\in\mathcal{N}} s_i p_i, \tag{6}$$

where

$$q_{ij}(p_i - p_j) = \min_{x_{ij}} \{(a_{ij} + p_j - p_i)x_{ij} | b_{ij} \le x_{ij} \le c_{ij}\}. \tag{7}$$

We henceforth refer to (LNF) as the *primal problem*, and note that standard duality results (see e.g., [6, 15, 23, 25]) relate primal-optimal and dual-optimal solutions via the complementary slackness conditions, and imply that the optimal primal cost equals the optimal dual cost.

The special structure of the dual cost (6) motivates solution by Gauss-Seidel relaxation (or coordinate ascent methods). The idea is to choose a single node i and change its price p_i in a direction of improvement of the dual cost, while keeping the other prices unchanged. Unfortunately there is a fundamental problem; the dual cost q is nondifferentiable (piecewise linear), and the relaxation idea may encounter difficulty at some "corner points," where the dual cost cannot be improved by changing any single node price.

One way to overcome this difficulty is used in the *ϵ-relaxation* algorithm first proposed in [2, 3] (see also [6, 7, 13, 15, 18, 19]). The main idea in the ϵ-relaxation method is to allow a single price p_i to change even if this worsens the dual cost. When p_i is changed, however, it is set to within a given $\epsilon > 0$ of the price that maximizes the dual cost along the ith coordinate. For ϵ small enough, it can be shown that the algorithm approaches the optimal dual cost sufficiently accurately to yield a primal-optimal solution.

A similar concept is used in the auction algorithm for assignment problems ([1, 4, 6, 15]). However, while in the ϵ-relaxation algorithm there is at most one node price change per iteration, the auction algorithm can raise two node prices simultaneously. In particular, the price of an unassigned person is raised implicitly through a "bid" as this person is assigned to a "preferred" object, and then the price of this object is also raised. This simultaneous price rise is an important feature that, we believe, accounts for the practical effectiveness of the auction algorithm. Experiments show that the ϵ-relaxation method applied to the assignment problem, is on the average far slower than the auction algorithm.

The main contribution of this paper is the development and analysis of a general algorithm which extends the concept of the auction algorithm for assignment problems by combining a price increase of a node with price increases of several neighboring nodes. This general algorithm can form the basis for a broad variety of auction algorithms tailored to the structure of particular problems. We investigate the termination properties of the generic algorithm for both feasible and infeasible problems, and we discuss some of the associated worst-case complexity issues.

As special cases of the generic algorithm, we develop two new algorithms: one for the general minimum cost flow problem (LNF), called *network auction*, and another for the k node-disjoint shortest path problem. The latter problem contains as special cases the classical assignment and the shortest path problems. The new algorithm is similar in structure to the recently proposed auction algorithm for shortest paths [5, 6], in that it maintains paths that are contracted or extended at each iteration. However, it requires a positive ϵ, in contrast with the algorithm of [5, 6], which corresponds to $\epsilon = 0$. We provide computational results showing that our new k node-disjoint shortest path algorithm outperforms existing algorithms by a broad margin. This algorithm is also well-suited for parallelization (see [24] for a related algorithm).

We also develop worst-case complexity bounds for the performance of the network auction algorithm. The complexity analysis differs significantly from previous analyses of coordinate ascent methods such as [13, 18, 19] in that the network auction algorithm includes new classes of iterations (the irregular nonsaturating δ-pushes discussed in Section 5) which must be bounded. We also develop a variation of the *sweep implementation* ([2, 13]). Combined with appropriate scaling techniques such as cost scaling or ϵ-scaling ([1, 12, 13, 18, 19]) our complexity analysis yields a complexity bound of $O(N^3 \log(NC))$ running time for the network auction algorithm when implemented using simple

data structures. Although improved polynomial complexity bounds are possible (the best such bound for the ϵ-relaxation method is $O(NA \log(N) \log(NC))$ for an implementation that uses dynamic trees [19]), the more complex data structures required are often detrimental to practical performance.

As a special case of our complexity analysis, we obtain the $O(N^3 \log(NC))$ bound for our earlier auction algorithm for transportation problems [9]; no polynomial complexity analysis was available for this transportation algorithm. Furthermore, under the assumption that the feasible flow range of all arcs is [0, 1], we can show that the generic algorithm has an $O(NA \log(NC))$ running time, where A is the number of arcs. This bound applies in particular to the new k node-disjoint shortest path algorithm.

The rest of this paper is organized as follows: in the next section we formulate the generic auction algorithm and establish its validity. In Section 3, we develop the network auction algorithm for problem (LNF) based on the generic algorithm. In Section 4, we give the auction algorithm for the k node-disjoint shortest path problem, and we show that it is a special case of the network auction algorithm. Section 5 contains our complexity analysis. Finally, in Section 6 we present computational results for the k node-disjoint shortest path problem.

2. Basic operations and the generic algorithm

The algorithms of this paper maintain a price vector p, and a capacity-feasible flow vector x, such that x and p jointly satisfy a relaxed form of the usual complementary slackness conditions known as ϵ-complementary slackness (ϵ-CS for short). We say that (x, p) satisfies ϵ-CS if x is capacity-feasible and

$$x_{ij} < c_{ij} \Rightarrow p_i - p_j \leq a_{ij} + \epsilon \quad \forall (i, j) \in \mathcal{A}, \tag{8a}$$

$$b_{ji} < x_{ji} \Rightarrow p_i - p_j \leq -a_{ji} + \epsilon \quad \forall (j, i) \in \mathcal{A}. \tag{8b}$$

The usefulness of ϵ-CS is due in large measure to the following proposition. A proof may be found in [2, 6, 13, 15]. Note that the proposition relies on our assumption that the problem data are integer.

PROPOSITION 1. *If $\epsilon < 1/N$, x is feasible, and (x, p) satisfies ϵ-CS, then x is optimal for (LNF).*

We now define some terminology and computational operations that play a significant role in our algorithms. Each of these definitions assumes that (x, p) is a flow-price pair satisfying ϵ-CS, and will be used only in that context.

Definition 1. An arc (i, j) is said to be *ϵ^+-unblocked* if

$$p_i = p_j + a_{ij} + \epsilon, \quad x_{ij} < c_{ij}. \tag{9}$$

An arc (j, i) is said to be ϵ^--*unblocked* if

$$p_i = p_j - a_{ji} + \epsilon, \quad b_{ji} < x_{ji}. \tag{10}$$

The *push list* of a node i, denoted P_i, is the (possibly empty) set of arcs (i, j) that are ϵ^+-unblocked, denoted P_i^+, and the arcs (j, i) that are ϵ^--unblocked, denoted P_i^-.

In all our algorithms, flow is allowed to increase only along ϵ^+-unblocked arcs and is allowed to decrease only along ϵ^--unblocked arcs. The next definition specifies the type of flow changes considered.

Definition 2. For an arc (i, j) [or arc (j, i)] of the push list P_i of node i, let δ be a scalar such that $0 < \delta \leq c_{ij} - x_{ij}$ ($0 < \delta \leq x_{ji} - b_{ji}$, respectively). A *$\delta$-push at node i on arc* (i, j) [(j, i), respectively] consists of increasing the flow x_{ij} by δ (decreasing the flow x_{ji} by δ, respectively), while leaving all other flows, as well as the price vector unchanged. A *saturating push* of node i on arc (i, j) [arc (j, i), respectively] is a δ-push with $\delta = c_{ij} - x_{ij}$ ($\delta = x_{ji} - b_{ji}$, respectively).

The next operation consists of raising the prices of a subset of nodes by the maximum common increment γ that will not violate ϵ-CS.

Definition 3. A *price rise* of a nonempty, strict subset of nodes I (i.e., $I \neq \emptyset$, $I \neq \mathcal{N}$), consists of leaving unchanged the flow vector x and the prices of nodes not belonging to I, and of increasing the prices of the nodes in I by the amount γ given by

$$\gamma = \begin{cases} \min\{S^+ \cup S^-\}, & \text{if } S^+ \cup S^- \neq \emptyset, \\ 0, & \text{if } S^+ \cup S^- = \emptyset, \end{cases} \tag{11}$$

where S^+ and S^- are the sets of scalars given by

$$S^+ = \{p_j + a_{ij} + \epsilon - p_i | (i, j) \in \mathcal{A} \text{ such that } i \in I, j \notin I, x_{ij} < c_{ij}\}, \tag{12}$$

$$S^- = \{p_j - a_{ji} + \epsilon - p_i | (j, i) \in A \text{ such that } i \in I, j \notin I, x_{ji} > b_{ji}\}. \tag{13}$$

In the case where the subset I consists of a single node i, a price rise of the singleton set $\{i\}$ is also referred to as a *price rise of node i*. If the price increment γ of (11) is positive, the price rise is said to be *substantive* and if $\gamma = 0$, the price rise is said to be *trivial*. Every scalar in the sets S^+ and S^- of (12) and (13) is nonnegative by the ϵ-CS conditions (8a) and (8b), respectively, so we have $\gamma \geq 0$. A trivial price rise changes neither the flow vector nor the price vector; it is introduced to facilitate the presentation. Note that a price rise of a single node i is substantive if and only if the set $S^+ \cup S^-$ is nonempty but the push list of i is empty.

The generic algorithm to be described shortly consists of a sequence of δ-push, and price rise operations. The following lemma lists some properties of these operations that are important in the context of the algorithm.

LEMMA 1. *Let (x, p) be a flow-price pair satisfying ϵ-CS.*

(a) The flow-price pair obtained following a δ-push or a price rise operation satisfies ϵ-CS.

(b) Let I be a subset of nodes with positive total surplus, that is, $\sum_{i \in I} g_i > 0$. Then if the sets of scalars S^+ and S^- of (12) and (13) are empty, problem (LNF) is infeasible.

Proof. (a) By the definition of ϵ-CS, the flow of an ϵ^+-unblocked and an ϵ^--unblocked arc can have any value within the feasible flow range. Since a δ-push only changes the flow of an ϵ^+-unblocked or ϵ^--unblocked arc, it cannot result in violation of ϵ-CS. If p and p' are the price vectors before and after a price rise operation of a set I, respectively, we have that for all arcs (i, j) with $i \in I$, and $j \in I$ or with $i \notin I$ and $j \notin I$, the ϵ-CS condition (8) is satisfied by (x, p') since it is satisfied by (x, p) and we have $p_i - p_j = p'_i - p'_j$. For arcs (i, j) with $i \in I$, $j \notin I$ and $x_{ij} < c_{ij}$ we have, using (11) and (12),

$$p'_i - p'_j = p_i - p_j + \gamma \leq p_i - p_j + (p_j + a_{ij} + \epsilon - p_i) = a_{ij} + \epsilon,$$

so condition (8a) is satisfied. Similarly, using (11) and (13), it is seen that for all arcs (j, i) with $i \in I$, $j \notin I$ and $x_{ji} > b_{ji}$, condition (8b) is satisfied.

(b) Since the sets S^+ and S^- are empty,

$$x_{ij} = c_{ij}, \quad \forall\ (i, j) \in \mathcal{A} \text{ with } i \in I, j \notin I, \tag{14}$$

$$x_{ji} = b_{ji}, \quad \forall\ (i, j) \in \mathcal{A} \text{ with } i \in I, j \notin I. \tag{15}$$

Using the definition (3) of surplus, we have

$$0 < \sum_{i \in I} g_i = \sum_{i \in I} s_i - \sum_{\{(i,j) \in \mathcal{A} \mid i \in I, j \notin I\}} x_{ij} + \sum_{\{(j,i) \in \mathcal{A} \mid i \in I, j \notin I\}} x_{ji}, \tag{16}$$

and by combining (14)–(16), it follows that

$$0 < \sum_{i \in I} s_i - \sum_{\{(i,j) \in \mathcal{A} \mid i \in I, j \notin I\}} c_{ij} + \sum_{\{(j,i) \in \mathcal{A} \mid i \in I, j \notin I\}} b_{ji}.$$

For a feasible vector, s_i is equal to the divergence of i, so the above relation implies that the sum of the divergences of nodes in I exceeds the capacity of the cut $[I, \mathcal{N} - I]$, which is a contradiction. Therefore, the problem is infeasible. □

The generic algorithm

Suppose that problem (LNF) is feasible, and consider a pair (x, p) satisfying ϵ-CS. Suppose that for some node i we have $g_i > 0$. There are two possibilities:

(a) The push list of i is nonempty, in which case a δ-push at node i is possible.
(b) The push list of i is empty, in which case the set $S^+ \cup S^-$ corresponding to the set $I = \{i\}$ via (12) and (13) is nonempty, since the problem is feasible [cf. Lemma 1(b)]. Therefore, from (11)–(13), a price rise of node i will be substantive.

Thus, *if $g_i > 0$ for some i and the problem is feasible, then either a δ-push or a substantive price rise is possible at node i.* Furthermore, since following a price rise at a node i, the push list of i will be nonempty [cf. (11)–(13)], *for a feasible problem a δ-push is always possible at a node i with $g_i > 0$, possibly following a price rise at i.*

The preceding observations motivate a method, called *generic algorithm*, which uses a fixed positive value of ϵ, and starts with a pair (x, p) satisfying ϵ-CS. The algorithm terminates when $g_i \leq 0$ for all nodes i; otherwise it continues to perform iterations. Each iteration consists of a sequence of δ-pushes and price rises, including at least one δ-push, as described below.

Typical iteration of the generic algorithm

Perform in sequence and in any order a finite number of δ-pushes and price rises; there should be at least one δ-push but not necessarily at least one price rise. Furthermore:

(1) Each δ-push should be performed at some node i with $g_i > 0$, and the flow increment δ must satisfy $\delta \leq g_i$.
(2) Each price rise should be performed on a set I with $g_i \geq 0$ for all $i \in I$.

The price rise operations of the generic algorithm may involve several nodes; however, in the special case where only one node is involved in each price rise, the generic algorithm can be further specified to obtain the ϵ-relaxation method, as shown in [8]. Similarly, for assignment and transportation problems, the auction algorithms of [1, 4, 9] are also special cases of the generic algorithm; for a detailed discussion of these equivalences, the reader is referred to [8]. Note that the generic algorithm can be further specified to obtain many new variations of auction algorithms for different classes of LNF problems, as discussed in subsequent sections.

The following proposition establishes the validity of the generic algorithm.

PROPOSITION 2. *Assume that the minimum cost flow problem (LNF) is feasible. If the increment δ of each δ-push is integer, then the generic algorithm terminates with a pair (x, p) satisfying ϵ-CS. The flow vector x is feasible, and is optimal if $\epsilon < 1/N$.*

Proof. We first make the following observations.

(a) The algorithm preserves ϵ-CS; this is a consequence of Lemma 1.
(b) The prices of all nodes are monotonically nondecreasing during the algorithm.
(c) Once a node has nonnegative surplus, its surplus stays nonnegative thereafter. The reason is that a δ-push at a node i cannot drive the surplus of i below zero (since $\delta \leq g_i$), and cannot decrease the surplus of neighboring nodes.
(d) If at some time a node has negative surplus, its price must have never been increased up to that time, and must be equal to its initial price. This is a consequence of (c) above and of the assumption that only nodes with nonnegative surplus can be involved in a price rise.

Suppose, to arrive at a contradiction, that the algorithm does not terminate. Then, since there is at least one δ-push per iteration, an infinite number of δ-pushes must be performed at some node m on some arc (m, n) or some arc (n, m). For concreteness, assume it is arc (m, n); a similar argument applies if the arc is (n, m). Since for each δ-push, δ is integer, an infinite number of δ-pushes must also be performed at the opposite end node n of the arc (m, n). This means that arc (m, n) becomes alternately ϵ^+-unblocked with $g_m > 0$ and ϵ^--unblocked with $g_n > 0$ an infinite number of times, which implies that p_m and p_n must increase by amounts of at least 2ϵ an infinite number of times. Thus we have $p_m \to \infty$ and $p_n \to \infty$, while either $g_m > 0$ or $g_n > 0$ at the start of an infinite number of δ-pushes.

Let $\mathcal{N}^\infty$ be the set of nodes whose prices increase to ∞; this set includes the nodes m and n. To preserve ϵ-CS, we must have, after a sufficient number of iterations,

$$x_{ij} = c_{ij} \quad \text{for all } (i, j) \in \mathcal{A} \text{ with } i \in \mathcal{N}^\infty,\ j \notin \mathcal{N}^\infty, \tag{17}$$

$$x_{ji} = b_{ji} \quad \text{for all } (j, i) \in \mathcal{A} \text{ with } i \in \mathcal{N}^\infty,\ j \notin \mathcal{N}^\infty. \tag{18}$$

After some iteration, by (d) above, every node in $\mathcal{N}^\infty$ must have nonnegative surplus, so the sum of surpluses of the nodes in $\mathcal{N}^\infty$ must be positive at the start of the δ-pushes where either $g_m > 0$ or $g_n > 0$. It follows using the argument of the proof of Lemma 1(b) [cf. (14)–(16)] that

$$0 < \sum_{i \in \mathcal{N}^\infty} s_i - \sum_{\{(i,j) \in \mathcal{A} \mid i \in \mathcal{N}^\infty,\, j \notin \mathcal{N}^\infty\}} c_{ij} + \sum_{\{(j,i) \in \mathcal{A} \mid i \in \mathcal{N}^\infty,\, j \notin \mathcal{N}^\infty\}} b_{ji}.$$

For any feasible vector, the above relation implies that the sum of the divergences of nodes in $\mathcal{N}^\infty$ exceeds the capacity of the cut $[\mathcal{N}^\infty, \mathcal{N} - \mathcal{N}^\infty]$, which is impossible. It follows that there is no feasible flow vector, contradicting the hypothesis. Thus the algorithm must terminate. Since upon termination we have $g_i \leq 0$ for all i and the problem is assumed feasible, it follows that $g_i = 0$ for all i. Hence the final flow vector x is feasible and by (a) above it satisfies ϵ-CS together with the final p. By Proposition 1, if $\epsilon < 1/N$, x is optimal. □

The example of Figure 1 shows how the generic algorithm may never terminate

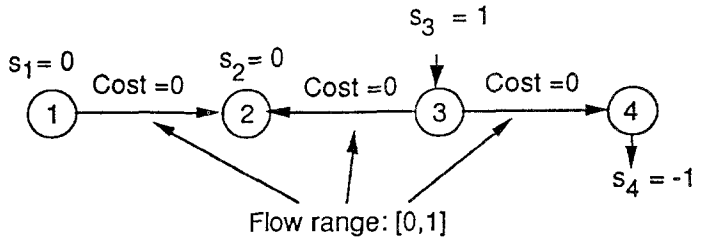


Figure 1. Example of a feasible problem where the generic algorithm does not terminate, if it does not perform at least one δ-push per iteration. Initially, all flows and prices are zero. Here, the first iteration raises the price of node 1 by ϵ. Subsequent iterations consist of a price rise of node 2 by an increment of 2ϵ followed by a price rise of node 1 by an increment of 2ϵ.

even for a feasible problem, if we do not require that it performs at least one δ-push per iteration.

Dealing with infeasibility

Let us consider now what happens when the problem is infeasible. Assume that the generic algorithm is operated so that for each δ-push, δ is integer. Then there are three possibilities:

(a) The algorithm terminates with $g_i \leq 0$ for all i and $g_i < 0$ for at least one i, in which case infeasibility is detected.
(b) The algorithm finds a subset of nodes I such that $\sum_{i \in I} g_i > 0$, and the sets of scalars S^+ and S^- of (12) and (13) are empty [cf. Lemma 1(b)], in which case infeasibility is again detected.
(c) The algorithm performs an infinite number of iterations and, consequently, an infinite number of δ-pushes.

In case (c), from the proof of Proposition 2 it can be seen that the prices of the nodes involved in an infinite number of δ-pushes will diverge to infinity. The following proposition gives a bound on the total price change of a node for a feasible problem. When this bound is violated, infeasibility is established.

PROPOSITION 3. *Suppose that the generic algorithm is applied to a feasible minimum cost flow problem with initial prices p_i^0. Then in the course of the algorithm, the price p_i of any node i with $g_i > 0$ satisfies*

$$p_i - p_i^0 \leq (N-1)(C+\epsilon) + \max_{j \in \mathcal{N}} p_j^0 - \min_{j \in \mathcal{N}} p_j^0, \tag{19}$$

where $C = \max_{(i,j) \in \mathcal{A}} |a_{ij}|$.

Proof. Let x^0 be a feasible flow vector and let (x, p) be a flow–price pair generated by the algorithm prior to its termination. Suppose that $g_i > 0$ for some i. Then by using the conformal realization theorem (see e.g. [6, 25]) on the flow vector $x - x^0$, we conclude that there exists a node s such that $g_s < 0$, and a simple path H starting at s and ending at i such that $x_{ij} - x_{ij}^0 > 0$ for all $(i, j) \in H^+$ and $x_{ij} - x_{ij}^0 < 0$ for all $(i, j) \in H^-$, where H^+ and H^- are the sets of forward and backward arcs of H, respectively. By ϵ-CS we have

$$p_j + a_{ij} \leq p_i + \epsilon, \quad \forall\ (i, j) \in H^+,$$
$$p_i \leq p_j + a_{ij} + \epsilon, \quad \forall\ (i, j) \in H^-.$$

Adding these conditions along H, we obtain

$$p_i - p_s \leq (N - 1)(C + \epsilon).$$

Since s has negative surplus, its price has not yet changed ($p_s = p_s^0$), so by subtracting p_i^0 from both sides of the above relation, we conclude that

$$p_i - p_i^0 \leq (N - 1)(C + \epsilon) + p_s^0 - p_i^0 \leq (N - 1)(C + \epsilon) + \max_{j \in \mathcal{N}} p_j^0 - \min_{j \in \mathcal{N}} p_j^0. \quad \square$$

The conclusion is that when the problem is feasible, the generic algorithm will terminate with a feasible x and a pair (x, p) satisfying ϵ-CS, as per Proposition 2, and when the problem is infeasible, the generic algorithm will detect infeasibility via one of the three tests (a)–(c) above, combined with the bound of (19).

An alternative way to deal with infeasibility is to introduce some artificial arcs to guarantee that the problem is feasible. Each artificial arc should have zero lower flow bound and high cost coefficient. The cost coefficient of each artificial arc should be high enough so that, for a feasible problem, its flow starts and stays at zero in the course of the algorithm. By using the bound of the preceding proposition, we can select the cost coefficients to be high enough so that in the case where the original problem is feasible, the artificial arcs never become ϵ^+-unblocked, and their flow stays at zero.

3. The network auction algorithm

The network auction algorithm described in this section is a particular variation of the generic algorithm. The algorithm starts an iteration from a node i with positive surplus and tries to exhaust the push list of i in preparation for a price rise. However, as it does so, it collects information from neighboring nodes that can be used to effect a price rise involving i and some of its neighbor nodes. The potential advantage here is that the corresponding price increment may be larger, thereby saving some iterations; furthermore, a price rise can be performed before the push list of i is exhausted.

To describe the typical iteration of the network auction algorithm, we need some definitions and a new operation. The *reject capacity* r_i of node i is defined as

$$r_i = \begin{cases} 0, & \text{if the push list } P_i \text{ is empty,} \\ \displaystyle\sum_{\{j|(i,j)\in P_i^+\}} (c_{ij} - x_{ij}) + \sum_{\{j|(j,i)\in P_i^-\}} (x_{ji} - b_{ji}), & \text{otherwise.} \end{cases} \tag{20}$$

Thus, r_i is the sum of the residual capacities of the arcs of the push list P_i.

Definition 4. A *reject operation at node* i consists of performing a saturating push on each of the arcs in the push list of i.

Note that in a reject operation at node i, the push list of i is emptied and the total amount of flow "pushed away" from i is equal to the reject capacity r_i.

The network auction iteration uses a subset L of neighbor nodes of i, which is empty at the start of the iteration. The nodes in L are the ones whose push list is emptied during the iteration through a reject operation. As a result, the prices of all nodes in L can be increased at the end of the iteration. This will occur regardless of whether the price of i is also increased. The price increase of the nodes in L, however, often has the beneficial effect of allowing a larger price increase for i than would otherwise be possible.

Typical iteration of the network auction algorithm

Step 0: (Select node) Select a node i with $g_i > 0$. If no such node exists, terminate the algorithm; else set $L = \emptyset$ and go to Step 1.

Step 1: (Select push list arc) Let P' be the set of arcs of the push list of i whose end node opposite to i does not belong to L. If P' is empty go to Step 3; else select an arc a from P' and go to Step 2.

Step 2: (δ-push) Let j be the end node of arc a, which is opposite to i. If $r_j \leq g_j$, perform a reject operation at node j, set $L := L \cup \{j\}$, and go to Step 1. Else let

$$\delta = \begin{cases} \min\{r_j - g_j, g_i, c_{ij} - x_{ij}\} & \text{if } a = (i, j), \\ \min\{r_j - g_j, g_i, x_{ji} - b_{ji}\} & \text{if } a = (j, i). \end{cases} \tag{21}$$

If $\delta = r_j - g_j$, perform a δ-push of i on arc a, perform a reject operation at node j, and set $L := L \cup \{j\}$; else just perform a δ-push of i on arc a. If as a result of these operations we obtain $g_i = 0$ go to Step 3; else go to Step 1.

Step 3: (Price rise) Perform a price rise of the set $\{i\} \cup L$. Then, if $L \neq \emptyset$, perform a price rise of L. Then, if $g_i = 0$ stop; else set $L = \emptyset$ and go to Step 1.

An alternative form of Step 3 is the following: Perform a price rise of the set $\{i\} \cup L$. Then, if $L \neq \emptyset$, sequentially perform a price rise of each of the nodes in L. Then, if $g_i = 0$ stop; else set $L = \emptyset$ and go to Step 1.

It can be shown that the above alternative form of Step 3 leads to larger price rises for transportation problems than the first form, because for bipartite graphs, there is no arc joining any pair of nodes in L. Therefore, the alternative form of Step 3 is preferable for bipartite problems, or more generally, in cases where for most iterations there is no arc connecting any two nodes of L.

We can show that the network auction algorithm is a special case of the generic algorithm. Indeed each iteration consists of δ-pushes, reject operations, and price rises, and the δ increments of all δ-pushes are positive integers. From (21) it is seen that $\delta \leq g_i$, while we have $r_j \leq g_j$ whenever a node j enters the set L and a reject operation is performed at j; this means that following a δ-push or a reject operation, the surplus of the corresponding node is nonnegative, so condition (1) of the generic algorithm is satisfied. Note also that the argument of the proof of Proposition 2 can be adapted to show that the number of δ-pushes per iteration is finite. Furthermore, since we have $g_i > 0$ at the start and $g_i = 0$ at the end of an iteration, it follows that at least one δ-push must occur before the iteration can stop.

Regarding price rises, we note that they involve nodes with nonnegative surplus, thereby satisfying condition (2) of the generic algorithm. To show that the number of substantive price rises per iteration is finite, note that with each substantive price rise, the reject capacity of either node i or a neighbor node of i (belonging to L) is increased by an integer amount. It follows that the number of substantive price rises per iteration cannot be infinite, since the reject capacity of each node is bounded and the number of δ-pushes per iteration is finite. Finally, regarding the number of trivial price rises per iteration, we note that the first price rise in Step 3 involving the set $\{i\} \cup L$ will be trivial only if the modified push list P' (cf. Step 1) is nonempty (the push lists of all nodes in L are empty following the reject operation in Step 2), in which case we must have $g_i = 0$ and the iteration will stop at that visit to Step 3. Therefore with each visit to Step 3 except at most one, there will be at least one substantive price rise. Since the number of substantive price rises is finite, it follows that the number of visits to Step 3 is finite, implying that the number of trivial price rises is also finite. Thus, the network auction algorithm is a special case of the generic algorithm and performs at least one δ-push per iteration. Therefore, Proposition 2 guarantees the termination of the algorithm with an optimal flow vector obtained if $\epsilon < 1/N$.

Note that if the first price rise involving the set $\{i\} \cup L$ in Step 3 is trivial and L is nonempty, the subsequent price rise in Step 3 (or price rises, if the alternative form of Step 3 is used) involving the set L will be substantive, since following the reject operation in Step 2, the push lists of all the nodes in L are empty. Thus, with each visit to Step 3 for which the set L nonempty, there is a price increase of all the nodes of L. Practical experience, as well as the complexity analysis of the next section, suggest that high frequency and large size of price

rises is a good performance indicator, so the extra work needed to compute the set L may be compensated by the associated extra price rises.

4. An algorithm for the k node-disjoint shortest path problem

In this section we consider a generalization of the single origin/single destination shortest path problem, where instead of a single path, we seek k node-disjoint paths that minimize a linear cost. An example is a three-dimensional assignment problem, involving the optimal choice of k disjoint ordered triplets, where the cost of a triplet (i, j, m) is separable of the form $a_{ij} + a_{jm}$. We derive a specialized version of the network auction algorithm for this problem. Note that in the literature the term *k shortest path problem* has been used somewhat differently; it refers to finding the shortest, second shortest, etc., up to kth shortest path between an origin and a destination [17].

Suppose that we are given a graph with node set $\mathcal{N}$, arc set $\mathcal{A}$, and integer arc costs a_{ij}. In this section, by a *path* P we mean either a single node i (in which case we say that P is a *trivial* path), or else a sequence of arcs $(i_1, i_2), (i_2, i_3), \ldots, (i_{m-1}, i_m)$. If the nodes $i_1, \ldots, i_m$ are distinct we say that the path is *simple*. We refer to i_1 as the *starting* node of P and to i_m as the *terminal* node of P; if P is trivial, its unique node is viewed as both the starting and the terminal node of P. The *cost of a nontrivial path* P is the sum of the costs of its arcs. By a *cycle* we mean a sequence of arcs $(i_1, i_2), (i_2, i_3), \ldots, (i_{m-1}, i_1)$. If the nodes $i_1, \ldots i_{m-1}$ are distinct we say that the cycle is *simple*.

Let s and t be given nodes called the *origin* and the *destination*, respectively. We assume that:

(a) s has no incoming arcs, t has no outgoing arcs, and (s, t) is not an arc. Furthermore, each node except for t has at least one outgoing arc. (These assumptions are convenient for stating the algorithm but do not involve a loss of generality.)
(b) The cost of each cycle is positive.

For a given positive integer k, we want to find k nontrivial simple paths $P_1, P_2, \ldots, P_k$ that start at s, terminate at t, and satisfy the following conditions:

(a) The paths are node-disjoint, that is, any pair of paths from the set $\{P_1, P_2, \ldots, P_k\}$ shares no node other than s and t.
(b) The sum of the costs of $P_1, P_2, \ldots, P_k$ is minimal.

It is possible to view this problem as a special case of the minimum cost flow problem (LNF) by replacing each node i other than s and t with two nodes i^+ and i^-, which are connected with a zero cost arc (i^+, i^-), and by replacing each arc (i, j) with the arc (i^-, j^+) of cost a_{ij}, as shown in Figure 2. All arcs

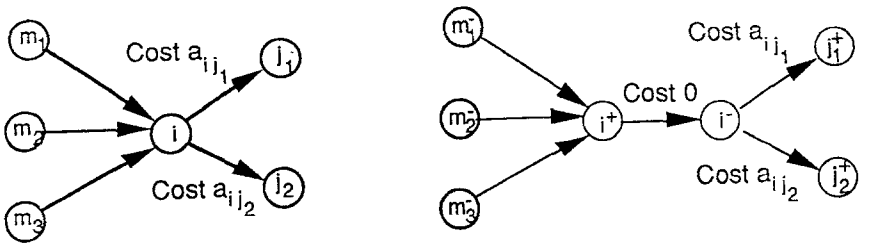


Figure 2. Converting the k node-disjoint shortest path problem to a minimum cost flow problem with all arcs having feasible flow range [0, 1]. Each node i is split into the two nodes i^+ and i^-, which are connected with a unit capacity and zero cost arc. Each arc (i, j) is replaced by an arc (i^-, j^+) of cost a_{ij}.

have feasible flow range [0, 1]. The supply of the origin is k, the supply of the destination is $-k$, and the supply of every other node is zero.

The following algorithm can be obtained by applying in a particular way the network auction algorithm to the above minimum cost flow problem. In particular, there will be price rises of pairs or triplets of nodes [either i^+ and i^-, or i^- and j^+ where (i, j) is an arc, or i^+, i^-, and j^+ where (i, j) is an arc]. These two-node or three-node price rises are almost as easy as single node price rises, and the algorithm is far more efficient than what would be obtained by straightforward use of the ϵ-relaxation method.

To simplify the presentation, we will describe the algorithm from first principles, and we will indicate more precisely the connections with the network auction algorithm later. We first introduce a price and flow vector structure, and a corresponding definition of ϵ-CS, which are adapted to the k node-disjoint shortest path problem. This form of ϵ-CS is somewhat more restrictive than the form given in Section 2.

ϵ-CS for the k node-disjoint shortest path problem

The subsequent k node-disjoint shortest path algorithm maintains the following:

(a) Two prices p_i^+ and p_i^- for each node $i \neq s, t$, which satisfy

$$p_i^- \leq p_i^+, \quad \forall i \neq s, t; \tag{22}$$

these prices correspond to the constituent nodes i^+ and i^- referred to earlier.

(b) A price p_s^- for the origin and a price p_t^+ for the destination, which are specified in terms of the remaining prices by the equations

$$p_s^- = \min\{z | z \geq a_{sj} + p_j^+ + \epsilon \text{ for } k \text{ or more arcs } (s, j)\}, \tag{23}$$

$$p_t^+ = \max\{z | a_{it} + z \leq p_i^- + \epsilon \text{ for } k \text{ or more arcs } (i, t)\}. \tag{24}$$

(c) A set of simple paths $P_1, \ldots, P_m$ and a set of simple cycles $C_1, \ldots, C_n$, which are all node-disjoint, and a flow vector x such that for all arcs (i, j)

$$x_{ij} = \begin{cases} 1 \text{ if } (i, j) \text{ belongs to one of the paths} \\ \quad P_1, \ldots, P_m \text{ or cycles } C_1, \ldots, C_n, \\ 0 \quad \text{otherwise.} \end{cases} \tag{25}$$

We require that out of the paths $P_1, \ldots, P_m$, exactly k are nontrivial and start at the origin, and at most k are nontrivial and terminate at the destination. Furthermore, a trivial path consisting of a single node, say i, belongs to the set $\{P_1, \ldots, P_m\}$ if and only if $i \neq s, t$, $p_i^- < p_i^+$, and no nontrivial path from $\{P_1, \ldots, P_m\}$ or cycle from $\{C_1, \ldots, C_n\}$ passes through i. (Note that the triplet (x, p^+, p^-) specifies completely the paths $P_1, \ldots, P_m$ and the cycles $C_1, \ldots, C_n$ based on the above requirements.)

We say that the triplet (x, p^+, p^-) *satisfies* ϵ-*CS* for the k node-disjoint shortest path problem if the above conditions hold and in addition

$$p_i^- \geq a_{ij} + p_j^+ - \epsilon, \quad \forall (i, j) \text{ such that } x_{ij} = 1, \tag{26a}$$

$$p_i^- \leq a_{ij} + p_j^+ + \epsilon, \quad \forall (i, j) \text{ such that } x_{ij} = 0. \tag{26b}$$

For a triplet (x, p^+, p^-) satisfying ϵ-CS, we say that one of the corresponding paths $P_1, \ldots, P_m$ is *active* if it terminates at a node other than the destination. Note that a trivial path consisting of a single node $i \neq s, t$ is active if and only if $p_i^- < p_i^+$. Note also that *if there are no active paths*, then in view of the requirement that out of the paths $P_1, \ldots, P_m$, exactly k are nontrivial and start at the origin, and no more than k are nontrivial and terminate at the destination, *the paths* $P_1, \ldots, P_m$ *must be* k *in number, must all start at* s, *and must all terminate at* t, *thereby yielding a feasible solution of the* k *node-disjoint shortest path problem.*

The following proposition gives the basis for the subsequent algorithm.

PROPOSITION 4. *Suppose that the triplet* (x, p^+, p^-) *satisfies* ϵ-*CS. Then if* $\epsilon < 1/N$, *there are no simple cycles corresponding to* (x, p^+, p^-). *If in addition none of the corresponding paths* $P_1, \ldots, P_m$ *is active, then these paths constitute an optimal solution of the* k *node-disjoint shortest path problem.*

Proof. If C is a simple cycle corresponding to (x, p^+, p^-), then for every arc (i, j) of C we must have $x_{ij} = 1$, and from (22) and (26),

$$p_i^+ \geq p_i^- \geq a_{ij} + p_j^+ - \epsilon.$$

By adding this relation over all arcs of C, we obtain

$$\text{Cost of } C = \sum_{(i,j) \in C} a_{ij} \leq (N - 1)\epsilon.$$

Since the arc costs are integer and $\epsilon < 1/N$, it follows that the cost of C is less or equal to zero, which contradicts our assumption that all cycle costs are positive.

If in addition there are no active paths, the vector x is a feasible solution that together with the price vector (p^+, p^-) satisfies ϵ-CS for the associated minimum cost flow problem, cf. Figure 2. The optimality proof for x is obtained by adapting the proof of Proposition 1 (see e.g., [6] or [15]) and by using the fact $p_i^+ \geq p_i^-$ for all $i \neq s, t$. We omit the details. □

The k node-disjoint shortest path algorithm starts each iteration with a triplet (x, p^+, p^-) satisfying ϵ-CS. The algorithm terminates if there is no active path. Otherwise, the algorithm selects an active path, and either *contracts* it by deleting its terminal node, or *extends* it by connecting its terminal node to another node; also the triplet (x, p^+, p^-) and at most one other of the corresponding paths and cycles are modified while maintaining ϵ-CS. As a result of the iteration, the path may get eliminated (if it consists of a single node or arc and is contracted) or may stop being active (if it joins a path that terminates at the destination). The number of active paths then decreases by one. It is also possible that the number of active paths stays the same as a result of the iteration.

To start the algorithm, we need an initial triplet (x, p^+, p^-) satisfying ϵ-CS. One way to obtain such a triplet is as follows:

Standard initialization

Set $x_{ij} = 0$ for all arcs (i, j), select p_i^+ arbitrarily for all $i \neq s$, set

$$p_i^- = \min \left\{ p_i^+, \min_{\{j \mid (i,j) \in \mathcal{A}\}} \{a_{ij} + p_j^+\} + \epsilon \right\}, \quad \forall\, i \neq s; \tag{27}$$

and set p_s^- and p_t^+ according to (23) and (24); then select k nodes j such that $(s, j) \in \mathcal{A}$ and $p_s^- \geq a_{sj} + p_j^+ + \epsilon$, and for all these nodes, set $x_{sj} = 1$ and $p_j^+ = p_s^- - a_{sj} + \epsilon$; then set $x_{it} = 1$ for all nodes arcs (i, t) with $p_i^- > a_{it} + p_t^+ + \epsilon$.

Contraction and extension operations

We now describe the operations of contraction and extension of an active path. Let (x, p^+, p^-) be a triplet satisfying ϵ-CS, and let $P_1, \ldots, P_m$ and $C_1, \ldots, C_n$ be the corresponding simple paths and cycles. Suppose that P is an active path with terminal node i.

A *contraction* operation for P can be performed in one of the following two circumstances:

(a) P consists of just node i and

$$p_i^+ < \min_{\{j|(i,j)\in\mathcal{A}\}} \{a_{ij} + p_j^+\} + \epsilon, \tag{28}$$

in which case the contraction consists of setting

$$p_i^+ = p_i^- = \min_{\{j|(i,j)\in\mathcal{A}\}} \{a_{ij} + p_j^+\} + \epsilon. \tag{29}$$

(In this case P is eliminated as a path.)

(b) P has a final arc, say (r, i), and

$$p_r^- - a_{ri} < \min_{\{j|(i,j)\in\mathcal{A}\}} \{a_{ij} + p_j^+\}, \tag{30}$$

in which case the contraction consists of setting

$$p_i^+ = p_i^- = \min_{\{j|(i,j)\in\mathcal{A}\}} \{a_{ij} + p_j^+\} + \epsilon,$$

deleting the final arc (r, i) from P, and setting $x_{ri} = 0$. If r is the origin node s, the following additional operations are executed: the price p_s^- is set to the value given by (23) (this value may be higher than the previous value of p_s^- since p_i^+ was just increased); also an arc (s, n) is found such that $x_{sn} = 0$ and $p_s^- = a_{sn} + p_n^+ + \epsilon$, and its flow x_{sn} is set to 1, while the flow of each incoming arc (r, n) with $r \neq s$ is set to zero. (This creates a new nontrivial path starting at the origin, to replace the path P consisting of the arc (s, i) that was eliminated through the contraction.) Following these changes, the price p_n^+ of each node n with $x_{sn} = 1$ is set to $p_s^- - a_{sn} + \epsilon$.

An *extension* operation for P is performed only if a contraction is not possible. Then we find a node j_i such that

$$j_i = \arg \min_{\{j|(i,j)\in\mathcal{A}\}} \{a_{ij} + p_j^+\},$$

and we also find

$$w_i = \begin{cases} \min_{\{j|j\neq j_i,(i,j)\in\mathcal{A}\}}\{a_{ij} + p_j^+\} + \epsilon & \text{if } i \text{ has two or more outgoing arcs,} \\ \infty & \text{if } (i, j_i) \text{ is the only outgoing arc from } i, \end{cases}$$

$$v_i = \begin{cases} p_r^- - a_{ri} + \epsilon & \text{if } P \text{ has a final arc } (r, i), \\ p_i^+ & \text{if } P \text{ consists of just node } i. \end{cases}$$

We then distinguish three cases, depending on whether j_i is the destination node, and whether an arc connecting the origin with j_i is part of a current path ($x_{sj_i} = 1$).

(a) If $j_i \neq t$ and $x_{sj_i} = 0$, the prices p_i^+ and p_i^- are set to

$$p_i^+ = v_i, \quad p_i^- = \min\{v_i, w_i\}.$$

Furthermore, the price $p_{j_i}^+$ is set to

$$p_{j_i}^+ = \min\{v_i, w_i\} - a_{ij_i} + \epsilon, \tag{31}$$

while the arc (i, j_i) is added to P and its flow is set to 1; also the flow of any incoming arc (n, j_i) with $n \neq i$ and $x_{nj_i} = 1$ is set to 0 (this could make n the terminal node of an active path).

(b) If $j_i \neq t$ and $x_{sj_i} = 1$, all the operations of the preceding case (a) are performed, including setting x_{sj_i} to 0. The following additional operations are then executed to create a new nontrivial path starting at the origin, replacing the path $P = (s, j_i)$ that was just eliminated [cf. case (b) of the contraction operation]: the price p_s^- is set to the value given by (23); also an arc (s, n) is found such that $x_{sn} = 0$ and $p_s^- = a_{sn} + p_n^+ + \epsilon$, and its flow x_{sn} is set to 1, while the flow of each incoming arc (r, n) with $r \neq s$ is set to zero. Following these changes, the price p_n^+ of each node n with $x_{sn} = 1$ is set to $p_s^- - a_{sn} + \epsilon$.

(c) If $j_i = t$, the prices p_i^+ and p_i^- are set to

$$p_i^+ = v_i, \quad p_i^- = \min\{v_i, w_i\},$$

and the arc (i, t) is added to P, while its flow is set to 1. If as a result, the number of paths terminating at t is $k+1$, the price p_t^+ is set to the value given by (24), and an arc (n, t) is found such that $x_{nt} = 1$ and $p_t^+ = p_n^- - a_{nt} + \epsilon$, and its flow x_{nt} is set to 0. [This eliminates a nontrivial path terminating at the destination, and since P was extended by arc (i, t), the number of nontrivial paths terminating at the destination is maintained at k.]

Note that in an extension operation it is possible that the extension node j_i is already part of P; then by setting $x_{ij_i} = 1$, a cycle C is obtained that consists of the portion of P between j_i and i and the arc (i, j_i). In this case, if j_i is the starting node of P, the active path P is replaced by the cycle C, and the number of active paths is reduced by one. Otherwise, the portion of P up to but not including j_i may become an active path.

By examining the nature of the contraction and extension operation, it is straightforward to verify the following:

(a) At the start of each iteration, the triplet (x, p^+, p^-) satisfies ϵ-CS.

(b) A contraction or extension that does not change the flow of any of the outgoing arcs from the origin is equivalent to an iteration of the network auction algorithm applied to the associated minimum cost flow problem described earlier.

(c) A contraction or extension that changes the flow of an outgoing arc from the origin [cf. case (b) of a contraction or case (b) of an extension] is equivalent

to two iterations of the network auction algorithm: an iteration starting at node i followed by an iteration starting at the origin.

k node-disjoint shortest path algorithm

Our algorithm starts each iteration with a triplet (x, p^+, p^-), and corresponding simple paths and cycles $P_1, \ldots, P_m, C_1, \ldots, C_n$ satisfying ϵ-CS.

Typical iteration of the k node-disjoint shortest path algorithm

Select an active path P. If no such path exists, terminate the algorithm; else if a contraction is possible for P [that is, if the corresponding condition (28) or (30) holds] perform the contraction, and otherwise perform an extension of P.

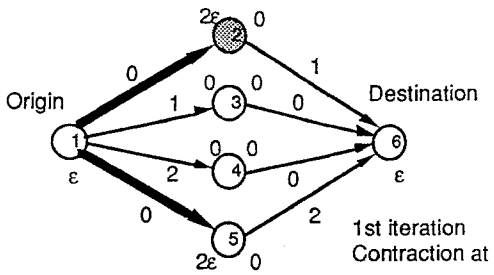
Figure 3 illustrates the algorithm for a simple example. From our earlier discussion, it is seen that the algorithm is a special case of the network auction algorithm. By using Proposition 2, it follows that for a feasible problem, the algorithm terminates, and by Proposition 4, the feasible solution obtained at termination is optimal if $\epsilon < 1/N$.

It is interesting to note that a $k \times k$ assignment problem can be converted to a k node-disjoint shortest path problem by adding an origin node s, which is connected with each person node with a zero cost arc, and by also adding a destination node t, which is connected to each object node with a zero cost arc. It can be verified that when the algorithm of this section is specialized to this problem, it becomes equivalent to the auction algorithm for the assignment problem.

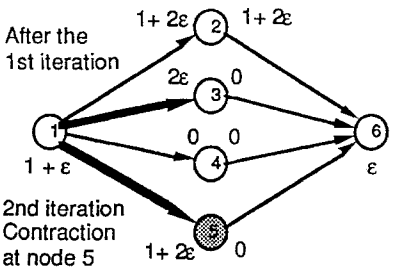
For another interesting connection, consider the case $k = 1$. Then the problem becomes a single origin/single destination shortest path problem. It can be verified that when the algorithm of this section is specialized to this problem but with the important difference that $\epsilon = 0$, it becomes equivalent to a recently proposed auction algorithm for shortest paths [5, 6].

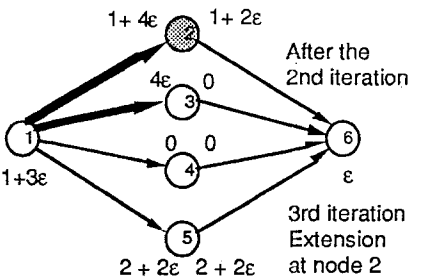
5. Complexity analysis

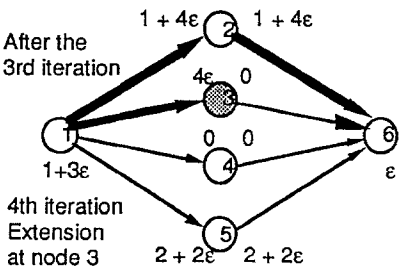
In this section, we derive a bound on the order of time required by a simple implementation of the network auction algorithm. Our analysis parallels a corresponding analysis of the ϵ-relaxation method given in [12, 13, 15], which in turn uses the sweep implementation ideas of [2] and some of the scaling analysis ideas of [18]. However, there are some novel and nontrivial features, such as the adaptation of the sweep implementation to the network auction algorithm, and, particularly, the distinction of nonsaturating δ-pushes into two types, regular and irregular. We concentrate on an unscaled version of the algorithm. Once the main complexity result for the unscaled algorithm (Proposition 5) is proved,



Problem data and initial conditions. Arc lengths are shown next to the arcs. Node prices are shown next to the nodes. Thick lines indicate arcs with flow equal to 1.







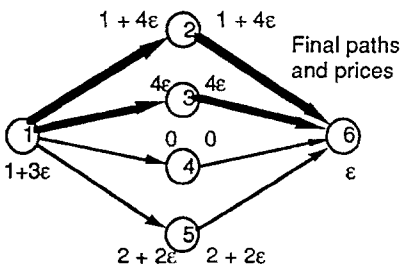


Figure 3. Illustration of the k node-disjoint shortest path algorithm for $k = 2$, starting with $p_i^+ = 0$ for all $i \neq s$ and using the standard initialization. The problem data is given in the first graph. The numbers on the left and the right sides of a node i are the prices p_i^+ and p_i^-, respectively. Thick (thin) line arcs are the ones with flow equal to 1 (0, respectively). Initially the active paths are (1, 2) and (1, 5). At the start of the second and third iterations there is only one active path, (1,5) and (1, 2), respectively.

the derivation of the corresponding results for scaled versions is straightforward, following established lines of analysis.

We first make some assumptions:

Assumption 1. Problem (LNF) is feasible.
Assumption 2. All arc cost coefficients are integer multiples of ϵ.
Assumption 3. All starting prices are integer multiples of ϵ, all starting flows are integer, and together they satisfy ϵ-CS.

We assume that the push lists of the nodes are maintained in a data structure that makes possible the addition and deletion of a single arc in $O(1)$ time; this is true, for example if each push list P_i is maintained in a doubly linked list. Then it is seen that selecting an arc in Step 1 takes $O(1)$ time, updating the push list of node i following a δ-push in Step 2 takes $O(1)$ time per δ-push, and updating the push list of a node i following a price rise involving node i in Step 3 takes $O(d_i)$ time per price rise and node, where d_i is the number of incident arcs of node i.

The admissible graph

A notion that is central in our analysis is the so-called *admissible graph*, introduced in [2], which consists of the push list arcs, except that the directions of those arcs that are incoming to the corresponding nodes are reversed to make them consistent with the direction in which flow is pushed in the network auction algorithm. Formally, the admissible graph is defined as $G^* = (\mathcal{N}, \mathcal{A}^*)$, where $\mathcal{A}^*$ contains arc (i, j) if either (i, j) is an ϵ^+-unblocked arc, or (j, i) is ϵ^--unblocked arc. Note that the admissible graph depends on the current pair (x, p) that satisfies ϵ-CS and changes as the pair (x, p) changes during the course of the algorithm. In particular, when there is a saturating push on an arc, the arc is removed from $\mathcal{A}^*$. However, δ-pushes cannot create any new arcs of the admissible graph. Furthermore, when there is a substantive price rise of a node set I in Step 3, all the arcs (i, j) and (j, i) with $i \in I$ and $j \notin I$ that belonged to $\mathcal{A}^*$ prior to the price rise are removed from $\mathcal{A}^*$, and an arc (i, j) is added to $\mathcal{A}^*$ if either an arc (i, j) (or (j, i)) with $i \in I$ and $j \notin I$ became ϵ^+-unblocked (or ϵ^--unblocked, respectively), as a result of the price rise. Thus following the price rise, there are no arcs (j, i) of the admissible graph that have a start node $j \notin I$ and an end node $i \in I$, leading to the conclusion that price rises cannot create any new cycles of the admissible graph (this will be shown more precisely in the proof of the subsequent Prop. 5). Our next assumption is that:

Assumption 4. Initially, the admissible graph has no arcs.

Assumption 4 can be satisfied by setting initially $x_{ij} = c_{ij}$ (or $x_{ij} = b_{ij}$) for

all arcs (i, j) with $p_i = p_j + a_{ij} + \epsilon$ ($p_i = p_j + a_{ij} - \epsilon$, respectively). Under this assumption, the admissible graph is initially acyclic and since, based on the preceding arguments, neither δ-pushes nor price rises can create a cycle, we conclude that *the admissible graph is acyclic throughout the algorithm.* (Again this will be shown formally as part of the proof of Proposition 5.)

The sweep implementation

In order to obtain the subsequent complexity bounds, we need a certain rule for choosing the starting node in each iteration. This rule is the basis for the sweep implementation referred to earlier, and uses an ordered list T of all the nodes, which is restructured repeatedly in the course of the algorithm. The initial choice of the list can be arbitrary. We say that node *i ranks higher* (or *lower*) than node j at some time, if the position of i in the list T is higher (or lower, respectively) than the one of j at that time. The order of nodes in the list will be shown to be related to the admissible graph (see the proof of the subsequent Proposition 5). In particular, it will be seen that a node i ranks higher than all nodes j such that there is a directed path from i to j in the admissible graph.

The order of nodes in T is changed only when there is a substantive price rise in Step 3. In particular, if the price rise involves a set I, the nodes of I are placed at the top of T in the order in which they appear in T prior to the price rise. The position of the nodes not in I is not changed. Figure 4 illustrates the rule for restructuring the list T following a price rise. We note that the restructuring of T can be done in $O(N)$ time per substantive price rise. In the case where the alternative form of Step 3 of the network auction algorithm or Step 3 of the ϵ-relaxation algorithm is used, the restructuring of T can be done more simply, in time $O(1)$ per single node price increase, by placing sequentially the nodes of L at the top of T as their price is increased. In practice one may want to maintain T in an appropriate data structure, such as a linked list, to minimize the restructuring overhead, but this is not necessary for the subsequent complexity bounds.

If a given iteration is started at node i, the list T is used to select the starting node i' for the next iteration as follows: Let N_i be the set of nodes that were ranking lower than i in T at the start of the given iteration and whose price did not change during the iteration. If N_i contains nodes that have positive surplus at the end of the iteration, then i' is chosen to be the highest ranking of these nodes; otherwise i' is chosen to be the highest ranking node in T among all the nodes that have positive surplus at the end of the iteration. (Thus, the algorithm goes down the list T selecting nodes with positive surplus and when it reaches the bottom of the list, it returns to the top of the list.)

A sequence of iterations between two successive times that the algorithm starts an iteration with the highest ranking node with positive surplus is called a *cycle.*

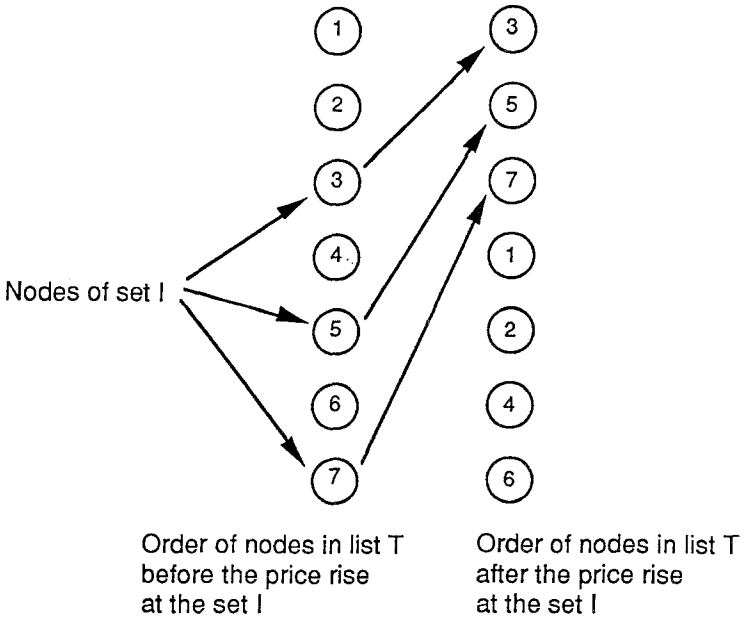


Figure 4. Illustration of the rule for restructuring the order of nodes in the list T following a price rise at the set I.

Note that the computation time for selecting the starting node of an iteration by going down the list T and checking for a positive surplus node, is $O(N)$ per cycle. Our final assumption is:

Assumption 5. The starting node of iterations of the network auction algorithm are chosen as described above.

Main result

We now introduce some terminology and state a lemma that is similar to one given for the ϵ-relaxation method in [12, 13, 15]. For any path H, we denote by $s(H)$ and $t(H)$ the start and end nodes of H, respectively, and by H^+ and H^- the sets of forward and backward arcs of H, respectively, as the path is traversed in the direction from $s(H)$ to $t(H)$. We say that the path is *simple* if it has no repeated nodes. For any price vector p and simple H, we define

$$d_H(p) = \max\left\{0, \sum_{(i,j)\in H^+}(p_i - p_j - a_{ij}) - \sum_{(i,j)\in H^-}(p_i - p_j - a_{ij})\right\}$$

$$= \max\left\{0,\ p_{s(H)} - p_{t(H)} - \sum_{(i,j)\in H^+} a_{ij} + \sum_{(i,j)\in H^-} a_{ij}\right\}. \tag{32}$$

It is seen that the following upper bound on $d_H(p)$ holds:

$$d_H(p) \leq p^+ - p^- + L, \tag{33}$$

where $p^+ = \max_i p_i$, $p^- = \min_i p_i$, and L is the maximum simple path length, where the length of each arc (i, j) is taken to be $|a_{ij}|$. Since any simple path can have at most $N - 1$ arcs, it is seen that when $p^+ - p^- = O(1)$, we have $d_H(p) = O(NC)$.

For any capacity-feasible flow vector x, we say that a simple path H is *unblocked with respect to* x if we have $x_{ij} < c_{ij}$ for all arcs $(i, j) \in H^+$ and we have $x_{ij} > b_{ij}$ for all arcs $(i, j) \in H^-$. For any price vector p and feasible flow vector x, denote

$$D(p, f) = \max\{d_H(p) | H \text{ is a simple unblocked path with respect to } x\}.$$

In the exceptional case where there is no simple unblocked path with respect to x, we define $D(p, f) = 0$. In this case, we must have $b_{ij} = c_{ij}$ for all (i, j) since any arc (i, j) with $b_{ij} < c_{ij}$ gives rise to a one-arc unblocked path with respect to x.

We have the following lemma:

LEMMA 2. *For every node, the total number of substantive price rises of a subset containing the node, up to termination of the network auction algorithm, is* $O(N + \beta(p^0)/\epsilon)$, *where* p^0 *is the initial price vector and*

$$\beta(p^0) = \min\{D(p^0, f) | x \text{ is feasible}\}. \tag{34}$$

The lemma has been proved for the ϵ-relaxation method in [12, 13, 15], and is based on showing that the relation

$$p_i - p_i^0 < (N - 1)\epsilon + \beta(p^0) \tag{35}$$

holds throughout the algorithm for all nodes i with $g_i > 0$. The proof for the network auction algorithm is essentially identical and will not be given; see also the proof of Proposition 3.

Our main complexity result is the following:

PROPOSITION 5. *Let Assumption 1–5 hold and let* p^0 *be the initial price vector. Then the network auction algorithm terminates in* $O(N^3 + N^2\beta(p^0)/\epsilon)$ *time.*

Proof. To economize on notation, we write β in place of $\beta(p^0)$. We will also need to distinguish between nonsaturating δ-pushes in Step 2 for which $\delta < r_j - g_j$ or $\delta = r_j - g_j$ in (21); these are called *regular* and *irregular* nonsaturating δ-pushes, respectively. Note that for each irregular δ-push, the node j of (21) enters the

set L and participates in a substantive price rise in the subsequent Step 3. The dominant computational requirements of the network auction algorithm are:

(1) For price rises and for updating the push list of nodes involved in the price rises.
(2) For restructuring the list T following price rises.
(3) For saturating δ-pushes.
(4) For irregular nonsaturating δ-pushes.
(5) For regular nonsaturating δ-pushes.
(6) For selecting a node i with $g_i > 0$ in Step 0.

There is also additional computation for updating the reject capacities of various nodes, but this work can be lumped into the work for δ-pushes and price rises, with no effect on the subsequently derived complexity bound.

We will show that the times required for the operations in (1)–(6) above can be estimated as follows:

For (1), $O(A(N + \beta/\epsilon))$.
For (2), $O(N^2(N + \beta/\epsilon))$; if the alternative form of Step 3 is used, the time required is $O(A(N + \beta/\epsilon))$.
For (3), $O(A(N + \beta/\epsilon))$.
For (4), $O(A(N + \beta/\epsilon))$.
For (5), $O(N^2(N + \beta/\epsilon))$.
For (6), $O(N^2(N + \beta/\epsilon))$.

Thus, we will obtain the desired $O(N^2(N + \beta/\epsilon))$ time bound.

Indeed, since by Lemma 2, there are $O(N + \beta/\epsilon)$ price increases for each node and a total of $O(N(N + \beta/\epsilon))$ price rises, the time required for (1) is $O(A(N + \beta/\epsilon))$ and the time required for (2) is $O(N^2(N + \beta/\epsilon))$. If the alternative form of Step 3 is used, then the restructuring of the list T can be done by placing sequentially the nodes of L at the top of T as their price is increased, so that the time required for (2) is $O(A(N + \beta/\epsilon))$.

Whenever an arc flow is set to either the upper or the lower bound due to a saturating push at one of the end nodes, it takes a price increase of at least 2ϵ by the opposite end node before the arc flow can change again. Therefore, there are $O(N + \beta/\epsilon)$ saturating pushes per arc. The computation time for each of these, including the time to remove the arc from the corresponding push list, is $O(1)$, so the time required for (3) is $O(A(N + \beta/\epsilon))$. Similarly, for each irregular nonsaturating δ-push there is a price rise of the corresponding node j that enters that set L. Thus there are $O(N + \beta/\epsilon)$ irregular nonsaturating pushes per arc, and the time required for (4) is $O(A(N + \beta/\epsilon))$.

There remains to estimate the computational requirements for (5) and (6). At this point, we will use the assumption that the algorithm is operated in cycles with the node order in each cycle determined by the list T. We will demonstrate

that the number of cycles up to termination is $O(N(N + \beta/\epsilon))$. Given this, we argue that for each cycle, there can be only one regular nonsaturating push per node in Step 2, for a total of $O(N^2(N + \beta/\epsilon))$ regular nonsaturating pushes. Since the time required for each of these pushes is $O(1)$, the time required for (5) is $O(N^2(N + \beta/\epsilon))$. Furthermore, the time to select a positive surplus node in Step 0 is $O(N)$ per cycle, so the time required for (6) is also $O(N^2(N + \beta/\epsilon))$. Thus the proof of the time estimates for the computations (1)–(6) stated above will be completed.

To show that the number of cycles up to termination is $O(N(N + \beta/\epsilon))$, we use the admissible graph $G^* = (\mathcal{N}, \mathcal{A}^*)$ and we argue as follows: a node i is called a *predecessor* of a node j if a directed path starting at i, ending at j, and having arcs oriented from i to j, exists in G^*. First we claim that immediately following a price rise of a node set I, there are no arcs (j, i) in $\mathcal{A}^*$ with $j \notin I$ and $i \in I$. To see this, note that if $(j, i) \in A$ with $j \notin I$ and $i \in I$ is ϵ^+-unblocked after the price rise, we must have $p_j > p_i + a_{ji} + \epsilon$ before the price rise, and, hence, $x_{ji} = c_{ji}$, implying that (i, j) is not in $\mathcal{A}^*$. The ϵ^--unblocked case is similar, establishing the claim. We next claim that G^* is always acyclic. This is true initially because, by Assumption 4, $\mathcal{A}^*$ is empty. δ-pushes can only remove arcs from $\mathcal{A}^*$, so G^* can acquire a cycle only immediately after a price rise of a node set I, and the cycle must include nodes of I as well as some nodes not in I. But since there are no arcs (j, i) with $j \notin I$ and $i \in I$ in the admissible graph following the price rise, no such cycle is possible. This establishes the second claim. Finally, we claim that the node list T maintained by the algorithm will always be compatible with the partial order induced by G^*, in the sense that every node will always appear in the list after all its predecessors. Again this is initially true because $\mathcal{A}^*$ starts out empty. Furthermore a δ-push does not create new predecessor relationships, while after a price rise of a node set I, there can be no predecessor of a node in I which does not belong to I, while the set I is moved to the top of the list before any possible descendants. This establishes the claim.

Let N^+ be the set of nodes with positive surplus that have no predecessor with positive surplus, and let N^0 be the set of nodes with nonpositive surplus that have no predecessor with positive surplus. Then, as long as no price rise takes place, all nodes in N^0 remain in N^0, and execution of an iteration starting at a node $i \in N^+$ moves i from N^+ to N^0. If there is no price rise during a cycle, then all nodes of N^+ will be added to N^0 by the end of the cycle, which implies that the algorithm terminates. Therefore, there will be a price rise during every cycle except possibly for the last one. Since the number of price increases per node is $O(N + \beta/\epsilon)$, there can be only $O(N(N + \beta/\epsilon))$ cycles.

The proof of the time estimates for (1)–(6) stated above is now complete and the desired overall time bound for the algorithm follows. □

Note that the classical max-flow problem can be formulated so that all arc costs a_{ij} are zero except for one arc cost which is unity ([15] p. 334), and with

an initial price vector p^0 such that $p^+ - p^- = O(1)$, we have $\beta(p^0) = O(1)$ (cf. (33)). By taking $\epsilon = 1/(N+1)$ in Proposition 5, it follows that the network auction algorithm solves the max-flow problem in $O(N^3)$ time.

Problems with unit arc capacities

When the feasible flow range of each arc is [0, 1], such as for example the assignment problem and the k node-disjoint shortest path problem, there are no regular nonsaturating pushes. For this reason, to obtain a good complexity bound, it is not necessary to maintain and restructure the list T as described earlier. Instead, a much simpler FIFO queue that includes the nodes with positive surplus can be used. With this algorithmic modification, the preceding analysis can be adapted to show that the complexity bound of Proposition 5 is reduced to $O(A(N + \beta(p^0)/\epsilon))$.

Complexity of the generic algorithm

Much of the preceding complexity analysis can also be applied to the generic algorithm under some broadly applicable assumptions. In particular, let us call a δ-push by node i *exhaustive* on arc (i, j) [or arc (j, i)] if $\delta = \min\{g_i, c_{ij} - x_{ij}\}$ [or $\delta = \min\{g_i, x_{ji} - b_{ji}\}$, respectively]. Let also n_i be the number of times that the price of node i is changed due to a price rise. Consider in addition to Assumptions 1–3, the following assumptions:

(a) The computation required for price rises is bounded by a constant times $\sum_{i \in \mathcal{N}} a_i n_i$, where a_i is the number of incident arcs of node i.
(b) Each δ-push requires $O(1)$ computation and the number of δ-pushes which are not exhaustive is bounded by a constant time $\sum_{i \in \mathcal{N}} n_i$.
(c) Between two successive price rises there can be at most N^2 exhaustive δ-pushes. (This assumption is satisfied in the network auction algorithm if the node selection policy is arbitrary but the algorithm is operated so that the admissible graph is acyclic.)

Then, for fixed ϵ, by using assumptions (a) and (b) above, we can show similar to the proof of Proposition 5 that the computation for price rises, saturating δ-pushes, and nonexhaustive δ-pushes is $O(A(N + \beta(p^0)/\epsilon))$. By using assumptions (a) and (c) above we can also show similar to the proof of Proposition 5 that the computation for nonsaturating δ-pushes is $O(N^3(N + \beta(p^0)/\epsilon))$. We thus obtain a $O(N^3(N + \beta(p^0)/\epsilon))$ bound for the algorithm. By exploiting the problem structure and by using data structures such as the ones of the sweep implementation, it may be possible to reduce the time bound for nonsaturating δ-pushes, which is the worst-case complexity bottleneck. Such data structures can be developed in

the context of particular algorithms, e.g. the network auction algorithm.

Scaled versions

We can consider also a scaled version of the network auction algorithm. Given Proposition 5, this analysis is virtually identical to the corresponding analysis of the ϵ-relaxation method given in the sources mentioned earlier. It can be found in our paper [8], which uses cost scaling. Here, we will just quote the main results. In particular, by using cost scaling as in [12] or [13], or ϵ-scaling as in [18] or [19], it can be shown based on Proposition 5 that the scaled version of the network auction algorithm with the sweep implementation as described earlier has an $O(N^3 \log(NC))$ running time, where $C = \max_{(i,j)\in A} |a_{ij}|$. Also, when the problem has unit arc capacities, we can obtain with a similar analysis an $O(NA \log(NC))$ bound. Finally, for the scaled version of the generic algorithm, we can show an $O(N^4 \log(NC))$ running time.

6. Computational results

In this section we present the results of some of our experimentation with the k node-disjoint shortest path algorithm of Section 4. The reader is also referred to several computational studies that have tested extensively auction algorithms for assignment and transportation problems [8, 9, 10, 11, 16].

We have implemented a code called AUCTION-KSP for k node-disjoint shortest path problems, which we tested against an implementation of the ϵ-relaxation method, called E-RELAX (given in [6]), the RELAXT-III code, which is an improved version of the one described in [14], and the primal-simplex code NETFLO, which is given in [20]. Figures 5 and 6 give some representative experimental results. NETFLO was slower by an order of magnitude than RELAXT-III and E-RELAX for the problems we tried, so its performance is not shown in these figures. AUCTION-KSP does not use scaling and this probably slows down its performance, particularly when k is relatively large. Despite this fact, AUCTION-KSP is uniformly and substantially faster than RELAXT-III and much faster than E-RELAX. This suggests that our specialized auction algorithm for the k node-disjoint shortest path problem is not just a heuristic improvement on the ϵ-relaxation method, but rather embodies some computational ideas that are genuinely interesting. We note also that the performance of AUCTION-KSP will probably improve substantially once we use scaling as well as "down iterations" where the prices of nodes with negative surplus are decreased. Down iterations have been shown to be very useful in the context of reverse auction for assignment problems [6, 11], and reverse auction for shortest path problems [5, 6].

We have also conducted much additional experimentation with the purpose to

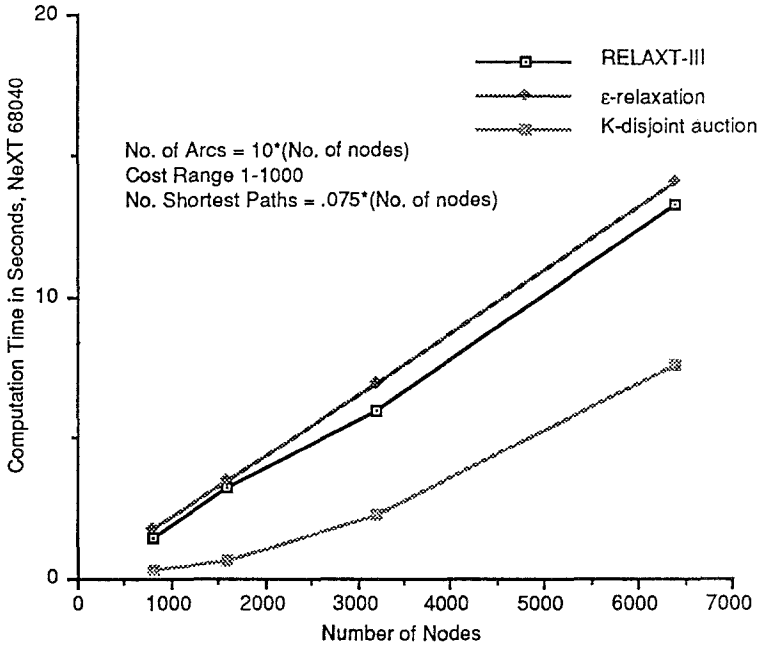


Figure 5. Comparison for the auction code AUCTION-KSP for k node-disjoint shortest path problems, with the transhipment codes E-RELAX, and RELAXT-III. Here the problems have a constant k while the number of nodes increases. The graphs of these problems were generated using NETGEN.

determine for what types of problems auction-like algorithms can form the basis for codes that outperform current state-of-the-art codes. This experimentation is not conclusive and cannot be presented here. However, the results seem to suggest that problems with a structure resembling the one of the assignment problem (bipartite or nearly bipartite structure, small and/or uniform sized supplies, small arc capacities) are good candidates for effective solution using specialized versions of the generic auction algorithm. Also a relatively simple problem structure such as the one of the max-flow, shortest path, and other related problems seems to favor the use of specialized auction algorithms.

References

1. D.P. Bertsekas, "A distributed algorithm for the assignment problem," Lab. for Information and Decision Systems Working Paper, Massachusetts Inst. Technol., Cambridge, MA, 1979.
2. D.P. Bertsekas, "Distributed asynchronous relaxation methods for linear network flow problems," LIDS Report P-1606, Massachusetts Inst. Technol., Cambridge, MA, 1986.
3. D.P. Bertsekas, "Distributed relaxation methods for linear network flow problems," in Proc. 25th IEEE Conf. on Decision and Control, Athens, Greece, 1986, pp. 2101–2106.

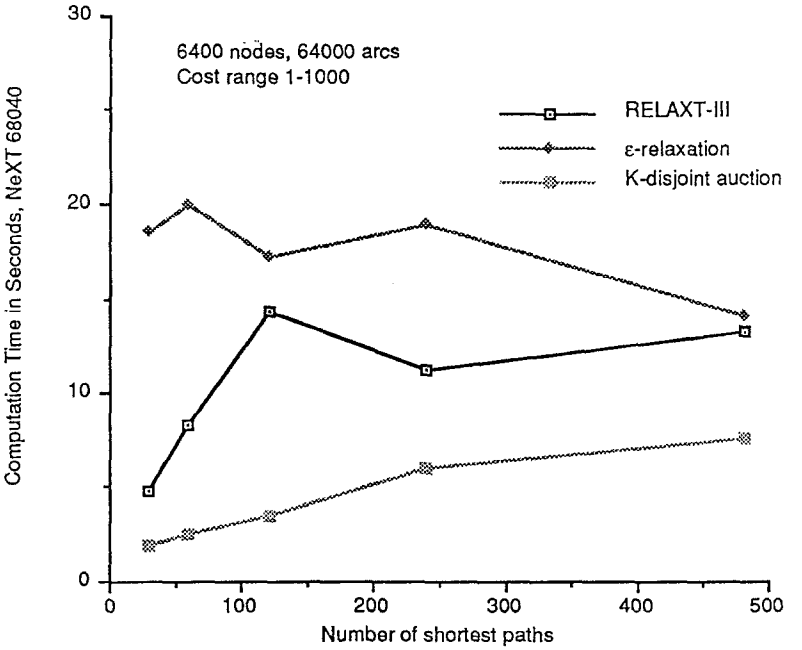


Figure 6. Comparison of the auction code AUCTION-KSP for k node-disjoint shortest path problems, with the minimum cost flow codes E-RELAX, and RELAXT-III. Here there is a single graph created by NETGEN[21] that has 6,400 nodes and 64,000 arcs, but the number of shortest paths k varies.

4. D.P. Bertsekas, "The auction algorithm: a distributed relaxation method for the assignment problem," Ann. Oper. Res., vol. 14, pp. 105–123, 1988.
5. D.P. Bertsekas, "The auction algorithm for shortest paths," SIAM J. on Optimization, vol. 1, pp. 425–447, 1991.
6. D.P. Bertsekas, Linear Network Optimization: Algorithms and Codes, MIT Press: Cambridge, MA, 1991.
7. D.P. Bertsekas, "Auction algorithms for network flow problems: a tutorial introduction," J. Comput. Optimization and Appl., vol. 1, pp. 7–66, 1992.
8. D.P. Bertsekas and D.A. Castañon, "The auction algorithm for the minimum cost network flow problem," Laboratory for Information and Decision Systems Report LIDS-P-1925, Massachusetts Inst. Technol., Cambridge, MA, 1989.
9. D.P. Bertsekas and D.A. Castañon, "The auction algorithm for transportation problems," Ann. Oper. Res., vol. 20, pp. 67–96, 1989.
10. D.P. Bertsekas and D.A. Castañon, "Parallel synchronous and asynchronous implementations of the auction algorithm," Parallel Comput., vol. 17, pp. 707–732, 1991.
11. D.P. Bertsekas, D.A. Castañon, and H. Tsaknakis, "Reverse auction and the solution of inequality constrained assignment problems," SIAM J. Optimization, vol. 3, pp. 268–297, 1993.
12. D.P. Bertsekas and J. Eckstein, "Distributed asynchronous relaxation methods for linear network flow problems," in Proc. of IFAC '87, Munich, Germany, July 1987.
13. D.P. Bertsekas and J. Eckstein, "Dual coordinate step methods for linear network flow problems," Math. Progr., Series B, vol. 42, pp. 203–243, 1988.

14. D.P. Bertsekas and P. Tseng, "RELAX: A computer code for minimum cost network flow problems," Ann. Oper. Res., vol. 13, pp. 127–190, 1988.
15. D.P. Bertsekas and J.N. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods, Prentice-Hall: Englewood Cliffs, NJ, 1989.
16. D.A. Castañon, "Reverse auction algorithms for assignment problems," in Network Flows and Matching, D.S. Johnson and C. McGeoch, eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 12, American Mathematical Society, pp. 407–430, 1993.
17. S.E. Dreyfus, "An appraisal of some shortest-path algorithms," Oper. Res., vol. 17, pp. 395–412, 1969.
18. A.V. Goldberg, "Efficient graph algorithms for sequential and parallel computers," Tech. Report TR-374, Laboratory for Computer Science, Massachusetts Inst. Technol., Cambridge, MA, 1987.
19. A.V. Goldberg and R. E. Tarjan, "Solving minimum cost flow problems by successive approximation," Math. Oper. Res., vol. 15, pp. 430–466, 1990.
20. J. Kennington and R. Helgason, Algorithms for Network Programming, Wiley: New York, 1980.
21. D. Klingman, A. Napier, and J. Stutz, "NETGEN—A program for generating large scale (un) capacitated assignment, transportation, and minimum cost flow network problems," Mgmt. Sci., vol. 20, pp. 814–822, 1974.
22. X. Li and S.A. Zenios, "Data parallel solutions of min-cost network flow problems using ϵ-relaxations," Report 1991-05-20, Dept. of Decision Sciences, The Wharton School, Univ. of Pennsylvania, Philadelphia, 1991.
23. C.H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall: Englewood Cliffs, NJ, 1982.
24. L.C. Polymenakos and D.P. Bertsekas, "Parallel shortest path auction algorithm," Laboratory for Information and Decision Systems Report LIDS-P-2151, Masachusetts Inst. Techno., Cambridge, MA, 1993; Parallel Computing (to appear).
25. R.T. Rockafellar, Network Flows and Monotropic Programming, Wiley-Interscience: New York, 1984.